STRUCTURAL INCONSISTENCY OF INTERNAL REPRESENTATIONS ON COGNITIVE PROCESSES OF SOFTWARE REUSE

Jinwoo Kim
School of Business Administration
Yonsei University
Seoul, 120-749, Korea

Phone: 02-361-2528
Fax: 02-361-5331
Email: jinwoo@bubble.yonsei.ac.kr

Abstract

This paper portrays software reuse as analogical mapping and rule development. In order to explore the impacts of inconsistent representations on the cognitive process of software reuse, it manipulates the internal representations of source and target problems and their program solutions. The experimental design for this study has two factors: 1) the degree of structural consistency between the source problem and the target problem, and 2) the
degree of structural consistency between the source problem and the source solution. A full-scale protocol analysis reveals that these two factors influence the cognitive process of mapping between source and target, the process of developing rules, and the interrelation between analogical mapping and rule development. The paper concludes by discussing the implications of these results for software reuse.

1. Introduction

Software reuse\(^1\) is currently being advanced as a solution to the problem of low productivity in the software industry (Krueger, 1992). The idea behind software reuse is that it should be faster to build software by recycling old programs than by building from scratch. However, the potential advantages of software reuse have not been

---

1) Software reuse includes recycling domain knowledge, requirement specifications, logical designs, program codes, and documentation. This paper focuses on the code reuse because the code reuse is closely related with other recycled items, as well as received enough credits for cognitive studies.
achieved in practice partially because of the cognitive difficulties of modifying existing programs to solve new problems (Woodfield, Embley, & Scott, 1987; Curtis, 1989).

Prior cognitive research in programming views software reuse as analogical problem solving (Maiden & Sutcliffe, 1992a, 1992b; Detienne, 1991; Hoc & Nguyen-Xuan, 1990). Holyoak and Thagard (1989b) have suggested that a core activity in analogical problem solving is mapping between the source analog (in software reuse, the old problem and its existing software solution) and the target analog (the new problem and the program to be written). This mapping is especially complex in software reuse because of the potential existence of inconsistent representations within the source analog (i.e., one representation for the source problem and a different representation for its software solution) as well as different representations between the source and target analogs (i.e., one representation for the source problem and a different representation for its target problem). However, there is little empirical evidence of the dynamic transformation among the inconsistent representations (Holyoak & Thagard, 1989a).

Besides analogical mapping, software reuse requires rule development (Novick and Holyoak, 1991; Holyoak, Novick, and Melz, in press; Nelson, Thagard, and Hardy, in press). We view computer programs as constructs for solving a set of instances, just as rule systems are intended to generate solutions for sets of similar instance problems. Consequently, we have characterized software design as the process of developing rules to solve a set of related
problem instances and transforming these rules into computer programs (Kim, Lerch and Simon, 1993). Therefore, analogical problem solving in software reuse has two inter-connected subprocesses: mapping between source and target, and rule development.

The unifying element in our study for both analogical mapping and rule development resides in the internal representations held by the programmer during software reuse. Internal representation is a key element in analogical mapping because the mapping is based on how people represent the source and target analogs (Vosniadou and Ortony, 1989). Similarly, internal representation has been found to influence the rule development process in various tasks such as software development (Kim, Lerch and Simon, 1993) and scientific discovery (Klahr and Dunbar, 1988). It has also been argued that internal representation is important in program reuse. For example, Curtis (1989) argues that a good representation for the source solution is the key ingredient for successful reuse.

In this research, we investigate the impacts of different sets of representations on the cognitive processes of analogical mapping and rule development. We are especially interested in the cases where representations between the source problem and its solution are structurally inconsistent, as well as where representations between the source problem and the target problem are structurally inconsistent. In order to experimentally observe the impacts of inconsistent representations upon the interactions between the rule and analogical mapping process, we manipulate the representations
of the source and target analogs by using problem and solution isomorphs. Problem isomorphs are problems that are basically the same (have isomorphic search spaces) but are disguised with different wordings (Hayes and Simon, 1974). Solution isomorphs are computer programs that implement the same functionality with different programming representations. Therefore, we could have four different representations: two for the source analog (one for the problem and the other for its solution program) and two for the target (one for the problem and the other for the program).

In our study, subjects were asked first to understand a source analog (both a problem and its software solution). Then they were given a target problem and asked to write a computer program to solve this target problem by reusing the existing program. We elected to use object-oriented programs for our source and target solutions. The rationale behind this decision is that advocates of Object-Oriented Design (OOD) claim that OOD is well suited for software reuse (Booch, 1992; Rumbaugh, Blaha, Premerlani, Eddy and Lorensen, 1992). Also, the components of object-oriented programs (objects, attributes, relations and methods) closely match the structure of problem representations proposed in the cognitive science literature, as shown in the next Section.

The next section presents a more detailed explanation of the role of representation in software reuse. Section Three reviews prior work on analogical mapping and rule development. Section Four presents the empirical study. The results of this study are presented in Section Five; we make extensive use of the analysis of the verbal
protocols which indicate the existence of different mapping strategies among experimental groups. Section Five also discusses the different interactions between analogical mapping and rule development exhibited by the different experimental groups. Section Six presents our conclusions about the role of internal representations in analogical mapping for software reuse. This paper ends with the implications of our results to more broad issues, including software reuse and analogical problem solving.

2. Problem Representation and Software Reuse

We define problem representation in software reuse as a cognitive structure for the problem constructed by programmers at any given time during their problem solving process on the basis of their domain-related knowledge (adapted from Chi, Feltovich and Glaser, 1981). Problem representations include three components: entities, relations, and roles (operations on entities). These three components are used to build the basic elements of problem spaces: the initial state, the goal state, and the available operators to move from the initial state to the goal state (Newell and Simon, 1972).

Entities have two basic characteristics: existence and state. An entity exists by itself distinctively from other entities. The state of an entity encompasses current values of all of the properties of the entity.
These properties are called attributes (Gentner, 1983). Relations are physical or conceptual connections among entities. Relations can be characterized by their cardinality. The cardinality of a relation may be one-to-one, one-to-many, or many-to-many (Rumbaugh, et al., 1991). These relations form default hierarchies along which the entities in a problem are organized (Holland, Holyoak, Nisbett and Thagard, 1986). Roles are abstract descriptions of the behaviors of entities, which are referred to as operations in Newell and Simon (1972). A role describes how an entity acts and reacts, in terms of its state and its relations to other entities. The characteristics of an entity's roles are determined by the entities connected to this role, their states, and their relations to the given entity. A set of roles of an entity implicitly expresses the behaviors of the entity, which will be performed explicitly by a set of prescriptions (i.e., rules) linked to that entity.

We use these three components to describe the representation of the computer programs, as well as the problems, because Object-Oriented (OO) program constructs can easily be mapped into these three components (Kim, Lerch and Simon, 1993). OO programs consist of autonomous objects (i.e., entities in problem representation) that have their own data and behaviors. OO programs work by objects executing their own behaviors and sending messages to other objects. Objects have attributes (i.e., attributes in problem representation) and the value of these attributes determines their current states. Objects have relations (i.e., relations in problem representation) with other objects by sending and receiving messages. Finally, objects have methods.
The name and list of arguments (i.e., the signature in OO programs) of the methods (i.e., roles in problem representation) determine the behaviors of an object, while the actual code that implements the methods are the rules that specify the behaviors of the object.

One of the main characteristics of software reuse is that the software reuse postulates the independence of problem from solution in terms of their representations. Problem descriptions and computer programs use different languages (problem descriptions use the language of the problem itself; computer programs use the language of the programming constructs) and have different constraints (the problem description has constraints from the problem only, but the computer program has constraints from the problem and from the programming construct). Since computer programs use different languages and have different constraints, the representation of the computer program does not necessarily have to be the same as the representation of the problem. Figure 1 shows a schematic view of the different multiple representations in software reuse. Representations 1 and 2 in Figure 1 are the programmer’s representations for the source and target problems. These representations may be modified to facilitate the task of reusing software. Representation 3 is the representation of the computer program given to the programmer to reuse. Finally, Representation 4 is the representation of the target program to be constructed by the programmer during reuse. Therefore, reuse is the transformation of the computer program with Representation 3 into a computer program (Representation 4) that satisfies the requirements of the target problem. The programmer is expected to select different
mapping strategies during reuse depending on the relations among the three available representations (Reps 1, 2 and 3).

![Diagram of four different representations in software reuse.](image)

Figure 1. Four Different Representations in Software Reuse.

3. Analogical Mapping and Rule Development

3.1. Analogical Mapping

The number of candidate mappings between a source and a target problem representation is potentially unmanageable (Clement & Gentner, 1991). Therefore, humans limit the mapping search by
using various types of independent constraints (surface similarity by Gentner and Toupin, 1986; structural consistency by Clement and Gentner, 1991; systematicity by Spellman and Holyoak, 1992; pragmatic centrality by Holyoak and Thagard, 1989a). One of the important constraints is Structural constraint, which is based on the relations among entities in the representations. The structural constraint leads people to prefer one-to-one entity correspondences and a consistently mapped dependency structure between different analogs (Clement & Gentner, 1991). For example, if Iraq attacked Kuwait maps to Germany attacked Poland, then Iraq should map to Germany, Kuwait should map to Poland, and attack should map to attack (Spellman & Holyoak, 1992).

Prior research in analogical problem solving has manipulated the problem representations of the source and/or target analogs (Rep1 and Rep2 in Figure 1) to study how the structural constraint guide analogical mapping (Gick and Holyoak, 1983; Gentner & Toupin, 1986; Holyoak & Koh, 1987; Novick, 1988; Gholson, Eymard, Long, Morgan, & Leeming, 1988; Bassok & Holyoak, 1989; Ross 1989; Clement & Gentner, 1991; Spellman & Holyoak, 1992). They found that when two problems do not share structural features (different relations and roles between analogs), analogists may construct incorrect mappings and generate incorrect analogical inferences. For example, Maiden and Sutcliffe (1992a, 1992b) found that programmers made errors in reusing requirement specifications because they neglected structural differences.

Multiple representations have been suggested as a means of
overcoming the difficulties in structural inconsistency situations. Spiro, Feltovich, Coulson and Anderson, (1989) have proposed that multiple source analogs may help people to construct correct analogical mappings in complex domains, such as medical reasoning, in which situations are often of structurally inconsistent. In such cases, a new analog may convey a different perspective from those conveyed by earlier ones, and the different perspective may be important for gaining insights in complex domains by supplementing, correcting, alternating, or enhancing earlier representations. The availability of multiple representations or perspectives on a given problem is also a frequent prescription for effective problem solving. For example, Kaplan and Simon (1989) show the importance of multiple representations for solving insight problems. However, there is little empirical research on how multiple representations facilitate analogical mapping in complex domains such as software reuse.

In our study, we manipulate the structural inconsistency between the source and target analogs by using problem isomorphs. Subjects in the consistency condition were given a source and a target problem with similar relations and roles but dissimilar entities. On the other hand, subjects in the inconsistency condition were given problems with the same entities, but with dissimilar relations and roles among these entities. We also tested how multiple representations in the source analog may help to overcome the difficulties from structural inconsistency. Half of the inconsistency subjects were given a source problem and a source solution sharing the same representation and then were asked to
solve a target problem with a different representation. The other half were given a source problem and its software solution having inconsistent representation, thereby providing two different representations for the source analog.

3.2 Rule development and computer programs

In software reuse, analogical mapping is only one of the cognitive operations required to develop programs. In related work (Kim, Lerch and Simon, 1993), we have proposed a cognitive framework in which software development is characterized as rule development. In this framework, rule development processes guide the analogical mapping between source and target programs, and then test and refine the analogously constructed rules. Our framework borrows mechanisms proposed in prior work on rule induction (Simon and Lea, 1977; Holland, Holyoak, Nisbett, and Thagard, 1986) and scientific discovery (Klahr and Dunbar, 1988; Langley, Simon, Bradshaw, and Zytkow, 1987). The framework is also closely related to our view of representations for problem descriptions and OO programs expressed in this paper. The framework defines rule development as a set of activities for transforming the implicit roles in the problem representation into explicit rule prescriptions. Therefore, roles determine WHAT entities do to solve the problem at a high level of abstraction. However, exactly HOW entities perform their roles is determined during the rule development process.

The framework posits the existence of two problem spaces for the
rule development process: a space of rules and a space of instances (Simon and Lea, 1977). It is the existence of these two problem spaces and the use of information drawn from one space to constrain search in the other space that distinguishes rule development from other forms of problem solving. Analogical mapping is one of the important activities in the rule space². By analogical mapping, programmers generate and refine new rules based on already existing rules (i.e., computer programs) for similar problems. In software design, cognitive activities in the instance space have been called mental simulations (Kant and Newell, 1984; Kim and Lerch, 1992). Programmers use mental simulations to test rules by generating instances and running the programs on these instances (Kant and Newell, 1984; Kim, Lerch and Simon, 1993). Schraagen (1993) has also found that mental simulation is one of the strategies that people use when mapping knowledge to a novel domain.

The interrelation between rule development and analogical mapping in software reuse is bi-directional. When programmers refine and evaluate mapped elements from the source solution using rule development, analogical mapping precedes rule development. Holyoak, Novick, and Melz (in press) have proposed that, for non isomorphic source and target analogs, mapping is followed by adaptation in which analogists perform general reasoning and memory search to evaluate the mapped rules. On the other hand,

² Other cognitive activities in the rule space are inductive and deductive inference. Refer to Kim(1993) for more details.
rule development may precede analogical mapping. In a simulation model developed by Nelson, Thagard, and Hardy (in press), rule development helps to shape the ways in which analogies are retrieved and mapped, while analogical mapping, in turn, helps to generate and refine rules. This rule development before analogical mapping is needed especially when a direct mapping is difficult to construct. In such situations, programmers need to develop rules for the target solution until they recognize the structural similarity that is hidden under misleading surface similarities.

Our study provides a detailed behavioral description of the interaction between rule development and analogical mapping. This is possible because program reuse requires intermingling these two processes. Moreover, most rule development activities are performed explicitly in software reuse because developing rules is the primary task in software design and software reuse. Therefore, subjects generate verbal utterances that explicitly specify the rules being developed so we do not have to infer indirectly when rules are generated (e.g., pauses in VanLehn, 1991) or the nature of these rules. We have observed programmers explicitly generating, refining and evaluating rules during software design (Kim, Lerch, and Simon, 1993). This characteristic of software design and reuse allows us to study the conditions under which analogists acquire rules about the target analog before mapping rules from the source analog.

4. Experiment
4.1. Subjects

Twenty-eight undergraduate students participated in the experiment for course credit. All students are seniors in the Math/Computer Science program at Carnegie Mellon University, and all of them were taking a software engineering course that taught an Object-Oriented design methodology (Rumbaugh et al. 1991). The software engineering class required the subjects to develop a real-world OO program (OOP) which consisted of 125 classes and more than 27,000 lines of code. The experiment was conducted almost at the end of their semester-long class (14 weeks). All of the subjects had taken at least four courses in programming and computer science, and all of them already had considerable experience with several programming languages and real-world software development projects.

4.2. Materials

4.2.1. Source Problems

The two source problems in this study are the Monster Change problem (3MC) and the Monster Transfer problem (3MT) as developed by Hayes and Simon (1974, 1976). The 3MC and the 3MT problems are Tower of Hanoi isomorphs disguised by different
wordings (Hayes and Simon, 1974). Both isomorphs were modified by stating that the computer program should solve any given initial configuration, rather than a single initial configuration as in the original formulations. Both isomorphs have three monsters and globes, and each monster tries to get a globe proportionate to its own size, starting from any given initial configuration of monsters and globes. In the 3MC problem, monsters change their globe size, while in the 3MT problem monsters transfer the globes from one monster to another. Both problems have three restrictions on the way to change or transfer the globes (e.g., Only one globe may be changed at a time [MC], Only one globe may be transferred at a time [MT]). Monsters try to get a globe to its own size without violating the three problem restrictions.

Figure 2 presents the expected initial problem representations (Rep 1 in Figure 1) for the 3MC and the 3MT problems. These representations are borrowed from Hayes and Simon (1974). Figure 2 shows that both problems have the same entities (monsters and globes). Both representations share the same HAVE relation but its cardinality is different between problems (in the 3MC problem the HAVE relation is one-to-one while this same relation is one-to-many for the 3MT problem). Both representations also share the NEXTSIZE relation that links monsters with each other in the order of their sizes. Finally, from previous studies (Hayes and Simon, 1974), it is expected that the main role will be allocated to the

3) Complete descriptions of the 3MC and 3MT problems are available from the author.
monster entities for both problems, although this main role is
different for each problem. In the 3MC problem we expect monsters
to be assigned the role of CHANGING the size of globes, while in the
3MT problem monsters are expected to be assigned the role of
TRANSFERRING globes. The representation for the 3MC has been
called the CHANGE representation, while that for the 3MT has been
called the TRANSFER representation (Hayes & Simon, 1974). Given
these two representations, activities for the two problems in the rule
space and the instance space will not be equivalent in cognitive
terms because these representations will induce different problem
The representation of the 3MT problem can be transformed to the
CHANGE representation by modifying the relations and the role as
follows: [relations: HAVE (globes, owner-monsters); NEXTSIZE
(globes, globes)] [ role: globes (CHANGING owner-monster size)].

<table>
<thead>
<tr>
<th>Entities</th>
<th>monsters, globes</th>
<th>monsters, globes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relations</td>
<td>HAVE (monsters, globes)</td>
<td>HAVE (monsters, globes)</td>
</tr>
<tr>
<td></td>
<td>NEXTSIZE (monsters, monsters)</td>
<td>NEXTSIZE (monsters, monsters)</td>
</tr>
<tr>
<td>Roles</td>
<td>monsters (CHANGING globe-size)</td>
<td>monsters (TRANSFERRING globes)</td>
</tr>
</tbody>
</table>

Figure 2. Representations for source problem (Rep1) for the 3MC and 3MT analogs.
4.2.2. Source Solutions

The source solutions for the 3MC and 3MT problems are Object-Oriented programs (OOP)4. Figure 3 presents the representations for the two source problems and their solutions. The problem representations in the left-hand column are for the 3MC source analog, and those in the right hand column are for the 3MT source analog. The two representations in the top row are based on source problems (Rep1 in Figure 1) and are the same as those shown in Figure 2. The representations for the source solution programs (Rep3 in Figure 1) are presented in the bottom row.

The source solution for the 3MC problem has the same CHANGE representation as the representation for the source problem (3MC). Monsters HAVE globes, there is a NEXTSIZE relation among monsters, and monsters have the role of CHANGING globe sizes. The source solution for the 3MC has an object Monster (class Monster), with a globe size attribute (int currentGlobeSize) to implement the HAVE (monsters, globes) relation. The NEXTSIZE relation is expressed by a pointer to the next larger monster (Monster * largerMonster). Finally, the role for the representation [monsters (CHANGING globe-size)] is implemented as [void changeGlobeSize(int desiredGlobeSize)] while the rules for performing this role are

4) Complete descriptions for the source solution programs for the 3MC and 3MT are available from the author.
specified in the method with the same name. In summary, both the source problem and its solution program for the 3MC analog have the same CHANGE representation. However, the representation for the solution program for the 3MT problem is different from the representation in the source problem of the 3MT problem. In the problem description (Rep1), Monsters HAVE globes, there is a NEXTSIZE relations among monsters, and monsters are assigned the role of TRANSFERRING globes. However, in the solution program (Rep3), the globes HAVE owner monsters, and there is a NEXTSIZE relation among globes. Moreover, the globes are assigned the role of CHANGING the size of their owner monsters.

Figure 3. Representations for Source Problem(Rep1) and Source Solution(Rep3) for the 3MC and 3MT analogs.
4.2.3. Target Problems

There are four target problems in this study, one for each experimental condition. Two problems are CHANGE problems (5MC and 5HC), and the other two are TRANSFER problems (5MT and 5HT). The 5MC and 5MT problems were generated by modifying the original 3MC and 3MT problems by adding more entities (from three to five). For example, the 5MT problem has three monsters and five globes. The 5HC and 5HT problem are based on the Tea Ceremony problem in Hayes and Simon (1974). These problems are also Tower of Hanoi isomorphs. The main change is that monsters are participants in a tea ceremony while globes are tasks to be performed by the participants. The 5HC and 5HT problems have the same structural representations as those for the 5MC and 5MT problems respectively.

4.3. Experimental Design

There are two experimental factors: structural consistency of the source solution and structural consistency of the target problem (as shown in Figure 4). Group CON-CON (CONsistent source solution, CONsistent target problem) and CON-IN (CONsistent source solution, INconsistent target problem) were given a 3MC problem for their source problem (Rep1), while Group IN-CON(INconsistent source solution, CONsistent target problem) and IN-IN(INconsistent source solution...
solution, Inconsistent target problem) were given a 3MT problem for their source problem. The two groups with the 3MC problem as their source problem (CON-CON and CON-IN) were given a solution program (Rep3) with the CHANGE representation. The two groups with the 3MT problem as their source problem (IN-CON and IN-IN) were given a solution program (Rep3) with a CHANGE representation of the TRANSFER problem (3MT), which is different from the representation in their source problem.

All four groups are required to make the same modifications to their source program. CON-CON and IN-CON groups need to make the following three changes: a) change monster to participant, b) change globe to task, and c) add two more participants. CON-IN and IN-IN groups also have to make three changes: a) change monster to globe, b) change globe to monster, and c) add two more monsters for IN-IN or two more globes for CON-IN.

![Figure 4. Analogs in Experimental Design.](image)
4.4. Procedure

Subjects were randomly assigned to one of the four groups. Ten subjects were allocated to each of the inconsistent target groups (CON-IN and IN-IN), while only four subjects were allocated to each of the consistent target group (CON-CON and IN-CON). The rationale behind this allocation is that we are mainly interested in the situations where analogical problem solving is difficult to achieve.

Subjects were given instructions about the general nature of the experiment and were told that verbal protocols would be collected. Subjects were trained in the thinking-aloud method using two traditional training tasks (Ericsson and Simon, 1984). The experimental sessions were divided into three phases: source-problem-understanding phase, source-solution-understanding phase, and target-solution-development phase. In the first phase (source-problem understanding) subjects were asked to read the source problems aloud and to solve two test cases. At the end of each of the two test cases, the experimenter presented to the subjects the most efficient solution for the current test-case. If differences were found between these two sets of moves, subjects were asked to understand the nature of these differences. These review sessions were intended to minimize the differences among subjects in terms of their level of source problem understanding, regardless of the quality of their own solutions. In the second phase, subjects were asked to understand their assigned C++ programs and
were told that they would be tested about the computer programs. After subjects stated they understood their program, they were asked to walk through the program with two test cases; then they were asked to answer several comprehension questions. The experimenter provided feedback to the subjects about their two walk throughs and their answers to the comprehension questions before they were given the target problem. Again, this feedback was intended to minimize differences among subjects. The feedback after the first and second phases were given to the subjects because we are mainly interested in the third phase, and therefore, need to control the differences of subjects in the first and second phases. In the third phase, subjects were given the target problem and asked to write a computer program. Subjects were explicitly told that they should use the existing programs. The experimental session continued until the subjects successfully finished all three phases or reached a given time limit (four hours). At the end of the experimental session, the experimenter asked subjects a set of questions about their prior knowledge in the TOH problem. It turned out that all subjects had been exposed to the TOH problem before the experiment, but did not remembered how to solve it in detail.

Of the twenty subjects allocated to the CON-IN and IN-IN groups, one subject per group was excluded from further analysis because they knew, before the experimental session, about the monsters problem being Tower of Hanoi isomorphs. Of the eight subjects allocated to the CON-CON and IN-CON groups, one subject per group was excluded because they did not have enough knowledge of
the programming language used in the source solutions (i.e., C++). In summary, there are 24 subjects in total, 9 for each of the inconsistent target problem groups (IN-IN and CON-IN) and 3 for each of the two consistent target problem groups (CON-CON and IN-CON).

4.5. Data Analysis

Detailed process data are needed to discover how the analogy is formed and used rather than what the end results look like. In software reuse, programmers may use analogies quite frequently, even though their final product may not include any analogously developed rules (Silverman, 1984). Also, time and accuracy, common measurements in problem-solving research, have been found to be only weakly related to the observed use of analogies (Novick, 1988; Novick & Holyoak, 1991).

All 24 verbal protocols were segmented into episodes. Episodes are small self-contained phases of highly organized activity (Newell and Simon, 1972). All episodes were classified into three categories: a) mental simulation, b) matching, and c) other. Mental simulation is the actual traversing of a solution path in the problem representation or running the solution program. Mental simulations are the major cognitive activity programmers use to understand computer programs (Detienne and Soloway, 1990; Pennington, 1987), as well as to develop programs (Kant and Newell, 1984; Guindon, 1990; Kim, Lerch and Simon, 1993). In matching episodes, subjects relate
multiple analogs. By closely examining mental simulations and matching episodes, we expect to distinguish among different mapping strategies as well as rule development processes. The Other category includes reading the problems, meta statements, and inductive and deductive inferences.

Episodes were classified as mental simulations if subjects traverse the problem either by making a move in the Tower of Hanoi isomorphs, or by running the program. Subjects may use pencil and paper while executing mental simulations. In that sense, the adjective "mental" is a little misleading; but this term will be kept in order to comply with current usage in computer programming and design. The schema for coding mental simulations is shown in Figure 5.

<table>
<thead>
<tr>
<th>Mental Simulation Number:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start time:</td>
</tr>
<tr>
<td>End time:</td>
</tr>
<tr>
<td>Logs:</td>
</tr>
<tr>
<td>Representation:</td>
</tr>
<tr>
<td>Entities:</td>
</tr>
<tr>
<td>Relations:</td>
</tr>
<tr>
<td>Roles:</td>
</tr>
</tbody>
</table>

*Figure 5. Coding Schema for Mental Simulation.*
The first three slots in the schema (Mental Simulation Number, Start time, and End time) are for record keeping. The Logs slot is used to keep track of the keywords and their timing. We pre-defined two sets of keywords that were expected to be used by subjects when running either the CHANGE or the TRANSFER representation. For example, "move" and "transfer" were identified as TRANSFER keywords, while "change" and "change-owner-monster" were identified as CHANGE keywords. We use the frequency and timing of these keywords to make inferences about the representations held by the subject while running the mental simulation. The possible values for the Representation slot are either CHANGE or TRANSFER. It is possible to have one or two Representation slots indicating that subjects have either one or two representations. Within the Representation slot, there are three sub-slots: Entities, Relations, and Roles.

Episodes are classified as matching when subjects attempt to relate more than one analog. In order to be classified as matching, episodes should include: 1) verbal utterances describing at least two different analogs, and 2) an indication that subjects are attempting to relate the analogs. Figure 6 shows the schema for the matching category. The first three slots are again for record keeping. The Type slot indicates the nature of the relating activity and has six possible values: 1) Listing, when subjects merely juxtapose some elements of the two analogs with no evaluation of their relations. 2) Identify-Similarity, when subjects explicitly stated that one analog was similar to the other analog, but without specifying which elements of one analog were similar to the elements of the other
analogs. 3) Identify-Difference, when subjects explicitly stated that the analogs were different in some dimension. 4) Mapping, when subjects explicitly stated that a specific element in one analog was equivalent to another element in the other analog. 5) Transform, when subjects converted one analog into the other after they had established the mappings between them. Finally, 6) No mapping, when subjects explicitly stated that two analogs have no common elements. The representations slot shows the elements (i.e., entities, relations, and roles) of both the source and target analogs. These sub-slots were coded using the same pre-defined sets of keywords as were used for coding mental simulations. The last slot, Mapped elements, indicates what elements were mapped. This slot may only have values if the value in the Type slot is either Mapping or Transform.

<table>
<thead>
<tr>
<th>Match Number :</th>
<th>Start time :</th>
<th>End time :</th>
<th>Type :</th>
</tr>
</thead>
<tbody>
<tr>
<td>Representations :</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source :</td>
<td>Target :</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Entities :</td>
<td>Entities :</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relations :</td>
<td>Relations :</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Roles :</td>
<td>Roles :</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mapped Elements :</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6. Coding Schema for Match Episode.
5. Results and Discussion

Our analysis focuses on the development of the solution for the target problem (Phase 3). Before presenting the mapping strategies for this phase, we briefly describe the results for the first two phases.

5.1 Phase 1 (Source problem understanding)

The entire protocols for the first phase were transcribed and checked for the occurrences of the predefined keywords of the CHANGE and TRANSFER representation. It turns out that subjects with the 3MC problem (CON-CON and CON-IN) had the keywords of CHANGE only, while subjects with the 3MT problem (IN-CON and IN-IN) had the keywords of TRANSFER only. These results indicate that subjects never deviated from their expected problem representations during the Phase 1. They also conform to prior studies in which subjects usually build the representations based on the given problem description (Hayes and Simon, 1976).

5.2 Phase 2 (Source solution understanding)

We coded all mental simulations for all subjects using the schema in
Figure 5 because mental simulations have been identified as the major cognitive activity in program comprehension (Detienne and Soloway, 1990; Pennington, 1987). The groups with the consistent source solution (CON-CON and CON-IN) subjects ran 3.2 mental simulations on average while the groups with the inconsistent source solution (IN-CON and IN-IN) subjects ran 6.8 on average. The coding of the mental simulations for CON-CON and CON-IN indicate that they always used the same CHANGE representation. On the other hand, IN-CON and IN-IN groups started running the computer program with the CHANGE representation, but then gradually moved towards the TRANSFER representation. In order to explore this gradual transformation of the representations, we evenly divided each verbal protocol for IN-CON and IN-IN subjects into 10 time intervals. We counted the number of keyword occurrences for both the CHANGE and the TRANSFER representations in each protocol interval. This is to show the percentage of TRANSFER keywords in each time interval combined for all subjects. For example, if there are 7 incidence of keywords for the TRANSFER representation and 3 incidence of keywords for the CHANGE representation in a given time interval, the percentage of the TRANSFER representation for that period becomes 70%. The resulting Figure 7 shows the gradual transformation of internal representation from the CHANGE to the TRANSFER.
These results indicate that subjects with inconsistent source solution (IN-CON and IN-IN subjects) initially used the CHANGE representation from the solution program. Then they gradually migrated to the TRANSFER representation given in the problem description. Subjects usually ran their first mental simulation with only the CHANGE representation, then they held both representations simultaneously as time progressed. The last mental
simulation was run mainly with the TRANSFER representation. This suggests that subjects with inconsistent source solution may have learned the two structurally inconsistent representations and may know how to map between them. At the very least, they are aware of the existence of multiple representations.

5.3. Phase 3 (Target solution development)

All four groups used different mapping strategies. The results for Phase 3 are organized into the analysis of the quantitative data and the analysis of the verbal protocols.

5.3.1. Quantitative Data

Macroanalysis of Mapping

All six subjects in the CON-CON and IN-CON groups (groups with structurally consistent target problem) reused the solution program successfully. Only 3 out of the 9 CON-IN subjects (group with the consistent source solution, but inconsistent target problem) were successful, while all the 9 IN-IN subjects (group with inconsistent source solution and inconsistent target problem) reused the solution program. We will refer to the three successful CON-IN subjects as Successful CON-IN, and to the other six CON-IN subjects as Unsuccessful CON-IN. The subjects with consistent target problem (CON-CON and IN-CON) spent less time reusing the program than
the successful subjects with inconsistent target problem (IN-IN and Successful CON-IN). CON-CON and IN-CON subjects spent 20.4 minutes on average while IN-IN and Successful CON-IN subjects spent 45.0 minutes. A one-way T-test comparing the time between consistent and inconsistent target problem showed that the time difference is significant below 0.001 probability ($t(16) = 4.13$, $p<.001$). Successful CON-IN subjects spent roughly the same time on average as IN-IN subjects (Successful CON-IN = 38.2 minutes; IN-IN = 47.27 minutes). Unsuccessful CON-IN subjects spent all the time left in the experimental session (102.8 minutes).

**Microanalysis of Mapping**

Figure 8 shows quantitative data for key events in Phase 3 for all five groups of subjects. First mapping is defined as the event in which subjects performed the first mapping between elements in the source analog and the target analog. This is the first episode that was coded as either Mapping or Transform in the Type slot of the Matching schema. The first row of Figure 8 shows the elapsed time before subjects finished their first mapping. CON-CON and IN-CON subjects completed their first mapping after 196 seconds and 131 seconds respectively. In contrast, groups with inconsistent target problem (Successful CON-IN, Unsuccessful CON-IN and IN-IN subjects) spent far more time completing their first mapping than groups with consistent target problems (IN-CON and CON-CON subjects) [$t(22) = 4.09$, $p<.001$].
**Class mapping** is defined as the Matching episode in which subjects found that in order to reuse the program they need to change the label of the object class in the program. For example, in the CON-IN group, class mapping is the episode in which subjects decided to have a Globe class in place of the Monster class in the computer program, and to change the names of the attributes and methods of the Monster class. The class mapping episode is used as the cut-off point in the analysis of the protocol because: 1) this is the essence of reusing the old program and mapping between source and target solutions, and 2) this event is easily identifiable in the protocols of successful subjects. Figure 8 shows that IN-CON subjects completed their class mapping after only 316 seconds. CON-CON subjects were a close second with 316 seconds. The IN-IN and Successful CON-IN took longer than the subjects with consistent target problems to complete the class mapping (IN-IN and Successful CON-IN: 938 seconds vs. CON-CON and IN-CON: 426 seconds; t(16) = 4.04, p<.001).
In summary, all subjects with the consistent target problem (CON-CON and IN-CON) reused the program successfully. They needed less time to map the existing program to their new problem. In contrast, only one third of CON-IN subjects reused the program, but all IN-IN did so. The quantitative results suggest that having an inconsistent solution for the source analog helps on during program reuse in the inconsistent target problem case. In order to understand the underlying cognitive mechanisms that generated these quantitative results, we performed a detailed protocol analysis of Phase 3. The focus of this analysis is in discovering differences in mapping strategies and the extent to which rule development guides this mapping among the experimental groups.

5.3.2. Protocol Analysis

This section presents the analysis of the verbal protocols of successful subjects until the completion of the class mapping or until the end of the experimental session for unsuccessful subjects. We have divided the analysis into four sections: one section for groups with the consistent target problem (CON-CON and IN-CON) and the other three for each group with the inconsistent target problem (Successful CON-IN, Unsuccessful CON-IN and IN-IN).

Protocol analysis for CON-CON and IN-CON groups

Figure 9 shows all the matching episodes coded for CON-CON and
IN-CON subjects using the coding schema shown in Figure 6. This figure indicates that the subjects mapped the elements between the source and target analogs by restriction matching. We define restriction matching as the use of the restrictions in the source and target problems for building one to one correspondences between their entities, relations and roles. Tower-of-Hanoi isomorphs have three restrictions referred to as Rst1, Rst2 and Rst3. For example, the Rst1 for the 3MT problem is, "1. Only one monster can transfer a globe at a time". The verbal protocols show that matching restrictions is sufficient for quickly achieving class mapping for these groups. Figure 9 shows that most matching episodes for these subjects were coded as Mapping in the Type slot. Mapping refers to subjects explicitly stating that one element in the source is equivalent to another element in the target. For CON-CON subjects the source is always 3MC (source problem) and the target is 5HC (target problem). Similarly, for IN-CON subject the source is always 3MT and the target 5HT. The last slot in the schema is Mapped Elements. All subjects have at least one matching episode, before class mapping, in which they matched restrictions between the source and the target. Nine (9) out of nineteen (19) matching episodes are restriction matching and no other type of matching occurs in all subjects.
<table>
<thead>
<tr>
<th>Subject</th>
<th>MatchNo.</th>
<th>Time (Sec)</th>
<th>Type</th>
<th>Source</th>
<th>Target</th>
<th>Mapped Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC1</td>
<td>Match1</td>
<td>40</td>
<td>Mapping</td>
<td>3MC</td>
<td>SHC</td>
<td>Rat2, Rat3</td>
</tr>
<tr>
<td></td>
<td>Match2</td>
<td>81</td>
<td>Mapping</td>
<td>3MC</td>
<td>SHC</td>
<td>Class Mapping</td>
</tr>
<tr>
<td>CC2</td>
<td>Match1</td>
<td>56</td>
<td>Identify-Similarity</td>
<td>3MC</td>
<td>SHC</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>Match2</td>
<td>77</td>
<td>Mapping</td>
<td>3MC</td>
<td>SHC</td>
<td>Rat1, Rat2, Rat3</td>
</tr>
<tr>
<td></td>
<td>Match3</td>
<td>71</td>
<td>Mapping</td>
<td>3MC</td>
<td>SHC</td>
<td>entities</td>
</tr>
<tr>
<td></td>
<td>Match4</td>
<td>131</td>
<td>Mapping</td>
<td>3MC</td>
<td>SHC</td>
<td>Class Mapping</td>
</tr>
<tr>
<td>CC3</td>
<td>Match1</td>
<td>112</td>
<td>Mapping</td>
<td>3MC</td>
<td>SHC</td>
<td>entities</td>
</tr>
<tr>
<td></td>
<td>Match2</td>
<td>85</td>
<td>Mapping</td>
<td>3MC</td>
<td>SHC</td>
<td>Rat1</td>
</tr>
<tr>
<td></td>
<td>Match3</td>
<td>178</td>
<td>Mapping</td>
<td>3MC</td>
<td>SHC</td>
<td>Rat2</td>
</tr>
<tr>
<td></td>
<td>Match4</td>
<td>90</td>
<td>Mapping</td>
<td>3MC</td>
<td>SHC</td>
<td>Rat3</td>
</tr>
<tr>
<td></td>
<td>Match5</td>
<td>91</td>
<td>Mapping</td>
<td>3MC</td>
<td>SHC</td>
<td>Relations</td>
</tr>
<tr>
<td></td>
<td>Match6</td>
<td>58</td>
<td>Identify-Difference</td>
<td>3MC</td>
<td>SHC</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>Match7</td>
<td>90</td>
<td>Mapping</td>
<td>3MC</td>
<td>SHC</td>
<td>Class Mapping</td>
</tr>
<tr>
<td>IC1</td>
<td>Match1</td>
<td>46</td>
<td>Mapping</td>
<td>3MT</td>
<td>SHT</td>
<td>entities</td>
</tr>
<tr>
<td></td>
<td>Match2</td>
<td>74</td>
<td>Mapping</td>
<td>3MT</td>
<td>SHT</td>
<td>Rat2</td>
</tr>
<tr>
<td></td>
<td>Match3</td>
<td>88</td>
<td>Mapping</td>
<td>3MT</td>
<td>SHT</td>
<td>Class Mapping</td>
</tr>
<tr>
<td>IC2</td>
<td>Match1</td>
<td>248</td>
<td>Mapping</td>
<td>3MT</td>
<td>SHT</td>
<td>Rat2, Rat3</td>
</tr>
<tr>
<td></td>
<td>Match2</td>
<td>134</td>
<td>Mapping</td>
<td>3MT</td>
<td>SHT</td>
<td>Rat2</td>
</tr>
<tr>
<td></td>
<td>Match3</td>
<td>38</td>
<td>Mapping</td>
<td>3MT</td>
<td>SHT</td>
<td>Class Mapping</td>
</tr>
<tr>
<td>IC3</td>
<td>Match1</td>
<td>99</td>
<td>Mapping</td>
<td>3MT</td>
<td>SHT</td>
<td>entities</td>
</tr>
<tr>
<td></td>
<td>Match2</td>
<td>100</td>
<td>Mapping</td>
<td>3MT</td>
<td>SHT</td>
<td>entities</td>
</tr>
<tr>
<td></td>
<td>Match3</td>
<td>24</td>
<td>Mapping</td>
<td>3MT</td>
<td>SHT</td>
<td>entities</td>
</tr>
<tr>
<td></td>
<td>Match4</td>
<td>78</td>
<td>Mapping</td>
<td>3MT</td>
<td>SHT</td>
<td>entities</td>
</tr>
<tr>
<td></td>
<td>Match5</td>
<td>88</td>
<td>Mapping</td>
<td>3MT</td>
<td>SHT</td>
<td>Rat2, Rat3</td>
</tr>
<tr>
<td></td>
<td>Match6</td>
<td>108</td>
<td>Mapping</td>
<td>3MT</td>
<td>SHT</td>
<td>Class Mapping</td>
</tr>
</tbody>
</table>

Figure 9. Matching Episodes for CON-CON and IN-CON subjects.

Figure 10 shows a simplified diagram for the restriction matching strategy. We will use similar diagrams to summarize the matching strategies for the other three groups without showing all the matching episodes in order to save space in the paper.
In terms of rule development, Tower of Hanoi solution isomorphs require three rule components: start-from-small, neither/nor, and recursive (Hayes and Simon, 1974). The start-from-small component refers to the need to first set the goal for the most constraint entity (the small monster in the 3MC problem). The neither/nor component specifies that, in order to accomplish the goal for a given entity, the program first needs to set up goals for removing all other blocking entities. Finally, the recursive component simply specifies that the removing procedure should be recursive. We analyzed to see if subjects expressed any of the three solution components while thinking aloud before they completed their class mapping. CON-CON and IN-CON subjects never stated knowledge about these three rule components for their target problem before class mapping. In fact, these subjects never ran mental simulations or developed rule components before mapping between analogs. This suggests that, for the structurally consistent target problem, knowledge about the rules for the target problem is
not required to successfully map the source solution to the target solution.

**Protocol analysis for Successful CON-IN group**

The protocol analysis for the matching episodes for the three successful CON-IN subjects indicates a very different mapping strategy from the groups with the consistent target problems. Successful CON-IN subjects had to discover the structural similarities between the source and the target analogs by using an intermediate schema. The analysis of the verbal protocols for the three Successful CON-IN subjects shows a remarkable degree of similarity in their mapping strategy. Figure 11 presents a schematic view of this mapping strategy.

![Diagram](image)

**Figure 11. Schematic View of Mapping Strategy for Successful CON-IN group**
The last step in their mapping strategy is restriction matching, the same strategy used by the other groups. However, successful CON-IN subjects performed many other steps before attempting restriction matching. Successful CON-IN subjects first generated an intermediate schema in the form of the INVERSE Tower of Hanoi (TOH) representation. They used this representation as a stepping stone between the 3MC (source problem) and the 5MT (target problem). The verbal protocols reveal that all three subjects performed three intermediate processes before matching problem restrictions. First, subjects expressed knowledge of the TOH. During this first process subjects only stated that either the target or source problems are similar to the INVERSE TOH problem (Identify-Similarity). We assume that their knowledge was specific enough so that they were able to discover that the monsters problems are INVERSE TOH isomorphs. Second, all three subjects proceeded to map the target problem (5MT) into the INVERSE TOH problem. All three subjects clearly stated that the 5MT is identical to the INVERSE TOH problem. They stated this fact while they were running mental simulations. Third, all three subjects then mapped the source problem (3MC) into the INVERSE TOH problem by also running mental simulations. Finally, after executing these two mappings using the intermediate schema, all three subjects mapped the source and target problems directly using the three restrictions. After these four processes were completed, all three subjects made their class mapping. The class mapping was followed by an "AHAI" episode (Kaplan and Simon, 1990).
Figure 12 shows all the mental simulations for SUCCESSFUL CON-IN subjects using the schema shown in Figure 7. Subjects spent approximately two thirds (68%) of their class mapping time running mental simulations. The remainder was spent in matching episodes, most of them occurring during mental simulations. The last column in Figure 12 shows the representations used in the mental simulations. If matching episodes were embedded in the mental simulation, then the value in this column shows two representations (e.g., MT <> TOH). If mental simulations were only used to run a single representation, then the value in this column is a single representation (e.g., MT). Figure 12 shows that all three subjects used mental simulations to map between MT and TOH, and between TOH and MC, except for SLS7. SLS7 mapped only between MT and TOH during phase 3, as shown in Figure 12. However, the analysis of his verbal protocol in phase 2 shows that he mapped TOH and MC during phase 2, so he did not have to do it in phase 3.

We speculate that SUCCESSFUL CON-IN subjects used the INVERSE TOH schema as a stepping stone in order to avoid the incorrect mapping induced by the semantic similarities between the two problems' entities (e.g., mapping monsters to monsters, and globes to globes). By using the INVERSE TOH schema, they were able to introduce semantically different entities (e.g., disks and pegs). Therefore, they can first map monsters to pegs, and then to globes. Likewise, globes are first mapped to disks, and then to monsters. In this way the intermediate schema is used to overcome the incorrect
mapping induced by surface similarities.

<table>
<thead>
<tr>
<th>Subject</th>
<th>MS No.</th>
<th>Time (sec)</th>
<th>Representations in mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLS1</td>
<td>MS1</td>
<td>108</td>
<td>TOM&lt;&gt; MT</td>
</tr>
<tr>
<td></td>
<td>MS2</td>
<td>796</td>
<td>TOM&lt;&gt; MT MC&lt;&gt; TOH</td>
</tr>
<tr>
<td>SLS7</td>
<td>MS1</td>
<td>661</td>
<td>MT</td>
</tr>
<tr>
<td></td>
<td>MS2</td>
<td>252</td>
<td>TOM&lt;&gt; MT</td>
</tr>
<tr>
<td>SLS10</td>
<td>MS1</td>
<td>193</td>
<td>TOM&lt;&gt; MT</td>
</tr>
<tr>
<td></td>
<td>MS2</td>
<td>121</td>
<td>TOM&lt;&gt; MT</td>
</tr>
<tr>
<td></td>
<td>MS3</td>
<td>52</td>
<td>MC&lt;&gt; TOH</td>
</tr>
<tr>
<td></td>
<td>MS4</td>
<td>42</td>
<td>TOH</td>
</tr>
<tr>
<td></td>
<td>MS5</td>
<td>108</td>
<td>MC&lt;&gt; TOH</td>
</tr>
<tr>
<td></td>
<td>MS6</td>
<td>224</td>
<td>MC&lt;&gt; TOH</td>
</tr>
</tbody>
</table>

Figure 12. Mental Simulation Episodes for successful CON-IN group.

In contrast to the subjects with the consistent target problem, all of the successful CON-IN subjects explicitly stated the three rule components for the target problem before they completed their class mapping. They developed most rules for the target problem before they reused the rules from the source solution. This suggests that mapping into the inconsistent target problem requires the development of rules for the target problem in order to discover the
structural similarities between the analogs.

In conclusion, the three successful CON-IN subjects used an intermediate schema to bridge between the source and target analogs. The intermediate schema helped them to overcome the incorrect mappings among semantically identical entities. At the same time, they developed enough knowledge about the target solution rules to discover the structural mapping between the two analogs.

Protocol data of the Unsuccessful CON-IN group

Protocol data for the six Unsuccessful CON-IN group subjects (ULS2, ULS3, ULS4, ULS5, ULS8, ULS9) were analyzed to uncover whether or not they performed the same three intermediate processes performed by Successful CON-IN subjects, before restriction matching. Figure 13 shows the summary of this analysis. None of the Unsuccessful CON-IN subjects performed all three intermediate processes. Four subjects retrieved prior knowledge about the TOH problem. Only two of them mapped the inverse TOH to the target problem (5MT). None of them mapped the INVERSE TOH to the source problem (3MC), and only one subject (ULS3) attempted using the three problem restrictions.

Four subjects (ULS2, ULS3, ULS5, and ULS9) decided not to reuse the source solution. They attempted to develop a new computer program instead of reusing the source solution program. None of them succeeded. The other two subjects (ULS4 and ULS8) modified
their mapping by correctly transforming the neither/nor rule component of the source program, but they were unable to extend this correct mapping to the other two rule components.

<table>
<thead>
<tr>
<th>Subject No.</th>
<th>TOH</th>
<th>INV-TOH to MT</th>
<th>INV-TOH to MC</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULS2</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>ULS3</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>ULS4</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>ULS5</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>ULS8</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>ULS9</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

Figure 13. Intermediate Processes for Unsuccessful CON-IN group.

Protocol data of the IN-IN group

The analysis of IN-IN verbal protocols indicates that IN-IN subjects used inconsistent source representations to directly map structural components (i.e., relations and roles) of the source solution to the target solution. IN-IN subjects used one of the two relations (i.e., NEXTSIZE) and the role (i.e., CHANGING) in the source solution to map the two analogs. Figure 14 depicts the mapping strategy of IN-
IN-IN subjects used the NEXTSIZE relation to map the semantically different entities in the two analogs (e.g., monsters to globes, and globes to monsters). They used this relation in two different ways. First, some of them transformed the largerGlobe attribute into the largerMonster attribute. A variant of this structural mapping is to transform the order of globe sizes into the order of monster sizes. This is equivalent to the prior transformation, but it includes additional information about the relationship among all of the entities of the same class. A different structural mapping is to directly use the role of the main entity (i.e., CHANGING). Seven IN-IN subjects transformed the role of changeOwnerMonster in the source program into changeGlobeSize for the target program.

The structural mappings performed by each IN-IN subject were analyzed, and it turned out that all of IN-IN subjects performed at
least one of the two types of structural mapping. Two subjects mapped only the NEXTSIZE relation, Two subjects mapped only the CHANGING role, and the rest five subjects performed both types of structural mapping.

It is also important to notice that mental simulations were not indispensable to achieve class mapping. In fact, only four (LM2, LM5, LM6 and LM8) out of nine subjects ran mental simulations before class mapping. In contrast to all other groups, IN-IN subjects did not use the problem restrictions to map between the two problem analogs. Instead, they directly mapped from the source solution to the target solution.

Figure 15 presents the rule components developed by IN-IN subjects before class mapping. Six IN-IN subjects had developed at least one rule component for the target solution (all three components by those who ran mental simulations, LM2, LM5, LM6, and LM8; one component by LM1 and LM4). On the other hand, three subjects (LM3, LM9, LM10) did not develop any rule component before class mapping. In order to understand further the differences between these two sub-groups within the IN-IN group (some-rule vs. no-rule), we compared the data for the two sub-groups during Phase 2. It turned out that the no-rule group had spent significantly more time in Phase 2 than the some-rule group (100.8 minutes for no-rule group, 41.1 minutes for some-rule group; t(7) = 1.96, p<.1). Furthermore, the no-rule group spent a greater proportion of Phase 2 running mental simulations with specific test-cases (68.9 % for the no-rule group, 37.2 % for the some-rule group; t(7) = 4.47, p<.005),
which are the same type of simulations run in Phase 3 by the four subjects in the some-rule group who developed all three rule components before mapping.

<table>
<thead>
<tr>
<th>Subject No.</th>
<th>Start-from-small</th>
<th>Neither/Nor</th>
<th>Recursive</th>
</tr>
</thead>
<tbody>
<tr>
<td>LM1</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>LM2</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>LM3</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>LM4</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>LM5</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>LM6</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>LM8</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>LM9</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>LM10</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

Figure 15. Knowledge about the Rules in the Target Problem for IN-IN group.

In conclusion, all IN-IN subjects directly mapped the structural components in the source solution to the target solution. There is a high variance among IN-IN subjects in terms of using mental simulations and developing rule components before class mapping. This high variance can be partially attributed to their activities in Phase 2. Some IN-IN subjects spent more time and ran more simulations in Phase 2 (the no-rule group); we speculate that this
allowed them to build more complete representations of the source analog and to recognize the structural similarities between the source and target analog. We also speculate that subjects who didn't do this extra work in Phase 2 (some-rule group) had to develop rule components for the target solution in order to discover the structural similarities between analogs.

6. General Discussion.

6.1. Summary of Results

6.1.1. Structural Consistency between the Source Problem and the Target Problem

Subjects with the structurally consistent target problem were all successful in reusing the computer program. They were also significantly faster than the successful subjects with inconsistent analogs (See section 5.3.1). They mapped between the source and target analogs by matching the three restrictions in the source and target problems (See Figure 10). Since the structural components (i.e., relations and roles) in the source and target problems are the same, they performed their class mapping by matching the entities (i.e., monsters to participants for CON-CON, and globes to tasks for
IN-CON) in the three restrictions. Also, they did not need to develop rules for the target solution before their class mapping, since structural similarity exists between source and target.

Providing inconsistent representations between the source problem and source solution did not have a significant effect on CON-CON and IN-CON. This may be the result of the small sample size for these two groups (three subjects per group). Alternatively, Gentner and Toupin (1988) found that expertise in mapping analogs (such as having inconsistent representations during program comprehension) had a greater impact on difficult analogical mappings (such as inconsistent target problems) than on simpler analogical mappings (such as consistent target problems). We may infer from these findings that, in software reuse, the increased cognitive effort of understanding inconsistent representations of the source solution during program comprehension (Phase 2) only pays off in situations where target problems are structurally different from the source problems. Therefore, our discussion of the role of inconsistent representations in analogical mapping is restricted to the two groups with inconsistent target problems.

6.1.2. Inconsistent Source Analogs for Inconsistent Target Problems

In the groups with inconsistent target problems, all (9/9) subjects with inconsistent source analogs (IN-IN) were able to reuse the computer program, while only one third (3/9) of subjects with a consistent representation (CON-IN) were able to map the source
program to the target program (See section 5.3.1). There are three possible explanations why the inconsistency between the source problem and source solution help programmers in mapping to the structurally inconsistent target problem.

One possible explanation is that the inconsistent source analog may help subjects to be aware of the possibility that the target problem is the same as the source problem, but with different structural representation. However, the analysis of the CON-IN verbal protocols showed that 7 out of nine CON-IN subjects retrieved the TOH problem (the 3 Successful plus 4 Unsuccessful CON-IN subjects in Figure 13). This suggests that CON-IN subjects are also aware that a change of representation is needed, just as IN-IN subjects are. Also, one of the other two unsuccessful CON-IN subjects attempted to map the neither/nor rule component between the two solution representations. Consequently, awareness of different representations by itself does not seem to explain the differences between single (CON-IN) vs. multiple (IN-IN) presentations groups.

Another possible explanation is that mapping experience acquired by IN-IN subjects during source solution understanding (Phase 2) helped them to overcome incorrect mappings. During Phase 2, IN-IN subjects were forced to switch the CHANGE representation in the source solution to the TRANSFER representation in the source problem (See Figure 7). In Phase 3 IN-IN subjects need to transform the CHANGE representation in the target problem to the TRANSFER representation in the source problem. Therefore, IN-IN subjects had
already acquired experience switching among representations. This may explain why IN-IN subjects did not seem surprised that the two problems are identical. In fact, we did not find an "AHA" episode for any of the IN-IN subjects in Phase 3. In contrast, all three Successful CON-IN subjects were clearly surprised to find out that the two solutions are identical to each other.

Finally, seven out of nine IN-IN subjects mapped between source and target solutions by using the CHANGING role shared by both solution representations (See Figure 14). Holoyak and Thagard (1989a) have proposed that analogical mapping is guided by the pragmatic centrality principle. They state that analogists prefer correspondences between elements that are judged to be sufficiently central so that a mapping should be found. In Object-Oriented software reuse, roles and their rules are central components for developing computer programs. Therefore, we believe that IN-IN subjects focused on the roles in the source and target solutions because of pragmatic centrality, and used the similarity of the two roles to successfully map the analogs.

5) The role in the source solution is not exactly the same as the role in the target problem for IN-IN subjects. The role in the source solution is "CHANGE-ownerMonsterSize," while the role in the target solution is "CHANGE-globeSize." However, this difference is not expected to be crucial in analogical reasoning. Holoyak and Thagard (1989) found that their ACME simulation model does not distinguish between mapping basic versions (CHANGE and CHANGE in this case) and mapping variant versions (CHANGE-ownerMonsterSize, CHANGE-globeSize). Gentner (1989) also found that mapping between FLOW-water and FLOW-electricity is not significantly different from the mapping between FLOW and FLOW.
Successful CON-IN subjects performed two intermediate mappings between the target and source problems using an intermediate schema (See Figure 11). Similar strategies have been observed or proposed in prior studies in analogical reasoning and problem representation. Bassok & Holyoak (1989) inferred that their subjects in a inter-domain transfer study were using an indirect remind-and-map process of transfer. This mechanism has been simulated in the computational model of analogical problem solving proposed by Holyoak and Thagard (1989b). Using an intermediate schema is similar to the multiple-analogs antidote against difficult mapping situations proposed by Spiro, Feltovich, Coulson and Anderson (1989). An intermediate representation provides an alternative representation for the source analog just as multiple analogies do. Our study shows clearly how people came up with an intermediate representation and how they construct step-by-step mapping between the original analogs and the newly made intermediate representation.

The use of intermediate representations in analogical mapping is also related to strategic re-representation processes. Moore and Newell (1974) developed a system that assumes humans are able to view a problem in several different ways when they try to map different problems. Michalski (1989) also proposed a representation system in which representations are viewed as dynamic structures built each time anew from a basic representation. The analysis of the Successful CON-IN verbal protocols empirically showed for the first time that complex analogical problem solving actually requires the interweaving of analogical mapping with strategic manipulations.
of representations.

6.2. Implications for Software Reuse

Our results clearly indicate that problem representation plays a key role in software reuse. Programmers may decide not to reuse software if they assess that the source and the target have substantially different representations (structurally inconsistent cases). In our study, Unsuccessful CON-IN subjects decided to stop attempting to reuse the old program after finding representational differences between their target problem and the source solution. Similar results have been found by Woodfield, Embley and Scott (1987). In their study, programmers stopped attempting to reuse software after perceiving that the differences between source and target were greater than a threshold value. Krueger (1992) argues that the cognitive distance between source and target representations is the major factor in deciding when to reuse. Our results show that multiple representations of the source analog are valuable in reducing this cognitive distance. Consequently, software reuse may be promoted by having programming environments that encourage building several different representations of existing programs. This suggestion is especially important when portraying class libraries provided by Object-Oriented Programming environments (e.g., Booch, 1992; Cox, 1987). Class libraries are sets of reusable components (e.g., a window class) that programmers are supposed to be able to plug into their programs when needed.
Our results suggest that a facility providing multiple different representations for class libraries should increase the likelihood of reuse.

Our results also indicate that mental simulation is a major mechanism programmers use to discover whether or not a reuse candidate is appropriate. The three Successful CON-IN subjects used mental simulations heavily to identify the mappings between the intermediate representation and their source and target representations. All IN-IN subjects used test case mental simulations in order to understand reuse candidates (either in Phase 2 or in Phase 3). We speculate that running mental simulations allows programmers to build a dynamic representation of problems: how problem states change when executing solution programs. Current programming tools (e.g., debuggers) provide information about the final representation or the interim states of the computer programs, but no information about changes in the states of the representation held by the programmer about the problem or its solution. It is possible that software reuse environments can be improved by helping programmers to run mental simulations. For example, reuse libraries may include the ability to run test cases and observe the dynamic changes in different representations. We believe that much of the difficulty in software reuse may stem from the fact that written code is impenetrable; it is difficult to read somebody else's code out of context and figure out what it may be run in mental simulations.

Obviously, our study does not deal with all of the variables
important in building environments to foster software reuse. Our study focused on just one important aspect of software reuse (i.e., given a source and target analog, how programmers map and adapt the source solution to the target solution), while software reuse in real life surely involves other steps in addition to this one. One important factor is the structure of the reward system in organizations. Programmers may be encouraged or discouraged to reuse software depending on how they are evaluated. Another important factor not dealt in our study is the issue of finding the appropriate reuse candidates in the first place. Little research has been conducted to study how developers decide whether or not a required software component already exists. Finally, our study focuses on the content structure of representations (e.g., relations and roles), but other aspects of representations are also important, such as mode of presentation (e.g., different programming languages).

The experimental methodology also limits our study in other respects. The experiment used a very small program, college students as programmers, and Tower of Hanoi isomorphs as problems. There is every reason to believe that problem representation and mental simulation are also key processes in software reuse on the scale of real-life problems, but our experimental studies do not predict the precise scale of these effects in complex situations. In the absence of substantial numbers of studies examining software reuse in naturalistic settings, experiments on smaller programs can give deep insights into the cognitive processes employed in software reuse, and can guide the
design of more complex studies. We are in the process of conducting further empirical studies to test our findings in real-life settings. Specifically, we are interested in assessing the importance of code representation and mental simulation in large projects.

ACKNOWLEDGMENTS

The research was supported by INI research grant #1-41354. The author appreciates comments on the earlier draft of this paper from Dr. F. Javier Lerch, Dr. Herbert, H. Simon, and two anonymous reviewers from Korean Journal of Cognitive Science.

REFERENCES


Approach. Reading, Massachusetts: Addison-Wesley.


design of algorithms. *Information processing and management*, 20, 97-118.


STRUCTURAL INCONSISTENCY OF INTERNAL REPRESENTATIONS ON COGNITIVE PROCESSES OF SOFTWARE REUSE

소프트웨어 재사용시의 인지과정상에 나타난 내적표상의 구조적 불일관성

김진우

이 글의 목적은 유추에 의한 사상(analogical mapping)과 규칙 계발(rule development)이란 측면에서 소프트웨어의 재사용이 가진 인지과정상의 특성을 밝히는 것이다.
이를 위해서 소프트웨어 재사용시 나타나는 비밀관적인 표상이 인지과정에 미치는 영향을 알아보기 위해서 원문제(source problems)와 표적문제(targent problems) 그리고 그들 문제의 해(solution)로서의 프로그램에 대한 내적 표상을 조작한다.
본 연구를 위한 실험은 두 가지 요인으로 구성된다. - 1) 원문제와 표적문제 사이의 구조적인 일관성의 정도(the degree of structural consistency), 2) 원문제와 그 해사이의 구조적인 일관성의 정도 - 프로토크출의 분석결과는 위의 두 용인이 원/표적사이의 사상(mapping)의 인지과정, 규칙계발과정. 그리고 유추에 의한 사상과 규칙계발사이의 상호 관계에 영향을 미치는 것을 알 수 있다.
마지막으로 소프트웨어 재사용에 대한 이상의 결과를 통해서 나타나는 시 준점이 가진 의미를 알아본다.

연세대학교
경영학과
서울 120-749, 대한민국

262